Technical
Instructions

IDC DOCUMENTATION

Retrieve Subsystem Software User Manual

Approved for public release;
distribution unlimited

Notice

This document was published July 2001 by the Monitoring Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. Every effort was made to ensure that the information in this document was accurate at the time of publication. However, information is subject to change.

Contributors

Michael Zatloukal, Science Applications International Corporation
Alexander Boresch, Science Applications International Corporation

Ordering Information

The ordering number for this document is SAIC-01/3007.

This document is cited within other IDC documents as [IDC6.5.20].

Retrieve Subsystem Software User Manual

CONTENTS

Retrieve Subsystem Software User Manual

FIGURES

Retrieve Subsystem Software User Manual

TABLES

About this Document

This chapter describes the organization and content of the document and includes the following topics:

- Purpose
- Scope
- Audience
- Related Information
- Using this Document

About this Document

PURPOSE

This document describes how to use the Retrieve Subsystem software of the International Data Centre (IDC). The software is a computer software component (CSC) of the Data Services Computer Software Configuration Item (CSCI) and is identified as follows:

Title:	Retrieve Subsystem

SCOPE

The manual includes instructions for setting up the software, using its features, and basic troubleshooting. This document does not describe the software's design or requirements. These topics are described in sources cited in "Related Information."

AUDIENCE

This document is intended for the first-time or occasional user of the software. However, more experienced users may find certain sections useful as a reference.

RELATED INFORMATION

The following documents complement this document:

- *Database Schema* [IDC5.1.1Rev2]
- *Configuration of PIDC Database* [IDC5.1.3Rev0.1]
- *Operations Manual for the IDC* [WGB98a]

See "References" on page 55 for a list of documents that supplement this document. The following UNIX manual (man) pages apply to the existing Retrieve Subsystem software:

- *dispatch*
- *MessageSend*
- *WaveAlert*

USING THIS DOCUMENT

This document is part of the overall documentation architecture for the IDC. It is part of the Technical Instructions category, which provides guidance for installing, operating, and maintaining the IDC systems. This document is organized as follows:

- Chapter 1: Introduction

  This chapter provides an overview of the software's capabilities, development, and operating environment.

- Chapter 2: Operational Procedures

  This chapter describes how to use the software and includes detailed procedures for startup and shutdown, basic and advanced features, security, and maintenance.

- Chapter 3: Troubleshooting

  This chapter describes how to identify and correct common problems related to the software.

- Chapter 4: Installation Procedures

  This chapter describes first how to prepare for installing the software, then how to install the executable files, configuration data files, database elements, and any Tuxedo files. It also describes how to initiate operation and how to validate the installation.

- References

  This section lists the sources cited in this document.

- Glossary

  This section defines the terms, abbreviations, and acronyms used in this document.

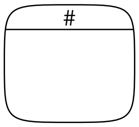
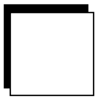
- Index

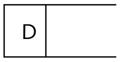
  This section lists topics and features provided in this document along with page numbers for reference.

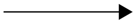
Conventions

This document uses a variety of conventions, which are described in the following tables. Table I shows the conventions for data flow diagrams. Table II lists typographical conventions.

TABLE I: DATA FLOW SYMBOLS

Description	Symbol[1]
process	
external source or sink of data	

data store
D = disk store
Db = database store
	

data flow	

[1] Most symbols in this table are based on Gane-Sarson conventions [Gan79].

TABLE II: TYPOGRAPHICAL CONVENTIONS

Element	Font	Example
database table	**bold**	**msgdisc**
database table and attribute, when written in the dot notation		**msgdest.***status*
database attributes	*italics*	*status*
processes, software units, and libraries		*MessageSend*
user-defined arguments and variables used in parameter (par) files or program command lines		`CMS_HOME=`*/cmss/rel*
titles of documents		*Continuous Data Subsystem*
computer code and output	`courier`	`#!/bin/sh`
filenames, directories, and websites		`process.par`
text that should be typed exactly as shown		`ls -l MessageSend*`

Chapter 1: Introduction

This chapter provides a general description of the software and includes the following topics:

- Software Overview
- Status of Development
- Functionality
- Inventory
- Environment and States of Operation

Chapter 1: Introduction

SOFTWARE OVERVIEW

The software of the IDC acquires time series and radionuclide data from stations of the International Monitoring System (IMS) and other locations. Time series data are passed through a number of automatic and interactive analysis stages, which culminate in the estimation of location and origin time of events (earthquakes, volcanic eruptions, and so on) in the earth, including its oceans and atmosphere. Radionuclide data are passed through spectra analysis to identify elements from volcanic or nuclear events. The results of the analysis are distributed to States Parties and other users by various means. Approximately one million lines of developmental software are spread across six CSCIs of the software architecture. Two additional CSCIs are devoted to non-developmental software and run-time data of the software.

The Data Services CSCI receives, archives, and distributes data through the following CSCs:

- Continuous Data Subsystem

  This software acquires time-series data according to a standard protocol and forwards the data to external users ([IDC3.4.2] and [IDC3.4.3]).

- Message Subsystem

  This software exchanges data in response to user requests. The data are formatted according to a standard protocol and exchanged through UNIX mail (see [IDC3.4.1Rev2]). This software also provides the interface to mail for the Retrieve and Subscription Subsystems.

- Retrieve Subsystem

  This software prepares messages, formatted according to the standard protocol, that retrieve segments of data from stations of the IMS auxiliary seismic network (see [IDC3.4.1Rev2]). The Message Subsystem software exchanges the messages and parses the response messages. The Retrieve Subsystem reconciles the received data with the requests.

- Subscription Subsystem

  This software maintains a subscriber database and prepares the regular data products for delivery to subscribers. The Message Subsystem receives the subscription requests and delivers the subscription products.

- Data Services Utilities and Libraries

  This software consists of utilities used by data services operators and libraries common to data services.

- Web Subsystem

  This software runs the IDC website.

- Authentication Services

  This software provides data signing and verification services to Data Services subsystems using the Digital Signature Algorithm (DSA).

Figure 1 shows the logical organization of the IDC software. IDC software is organized into seven catagories. The Retrieve Subsystem is one component of the Data Services CSCI. Data Services software provides the functionality to aquire, authenticate, store, and distribute data. The Retrieve Subsystem software uses the Message Subsystem and Authentication Services to acquire, store, and distribute auxiliary seismic data.

Figure 2 shows a processing flow model of the data requests and shows the relationship of the Retrieve Subsystem to other components of the system.

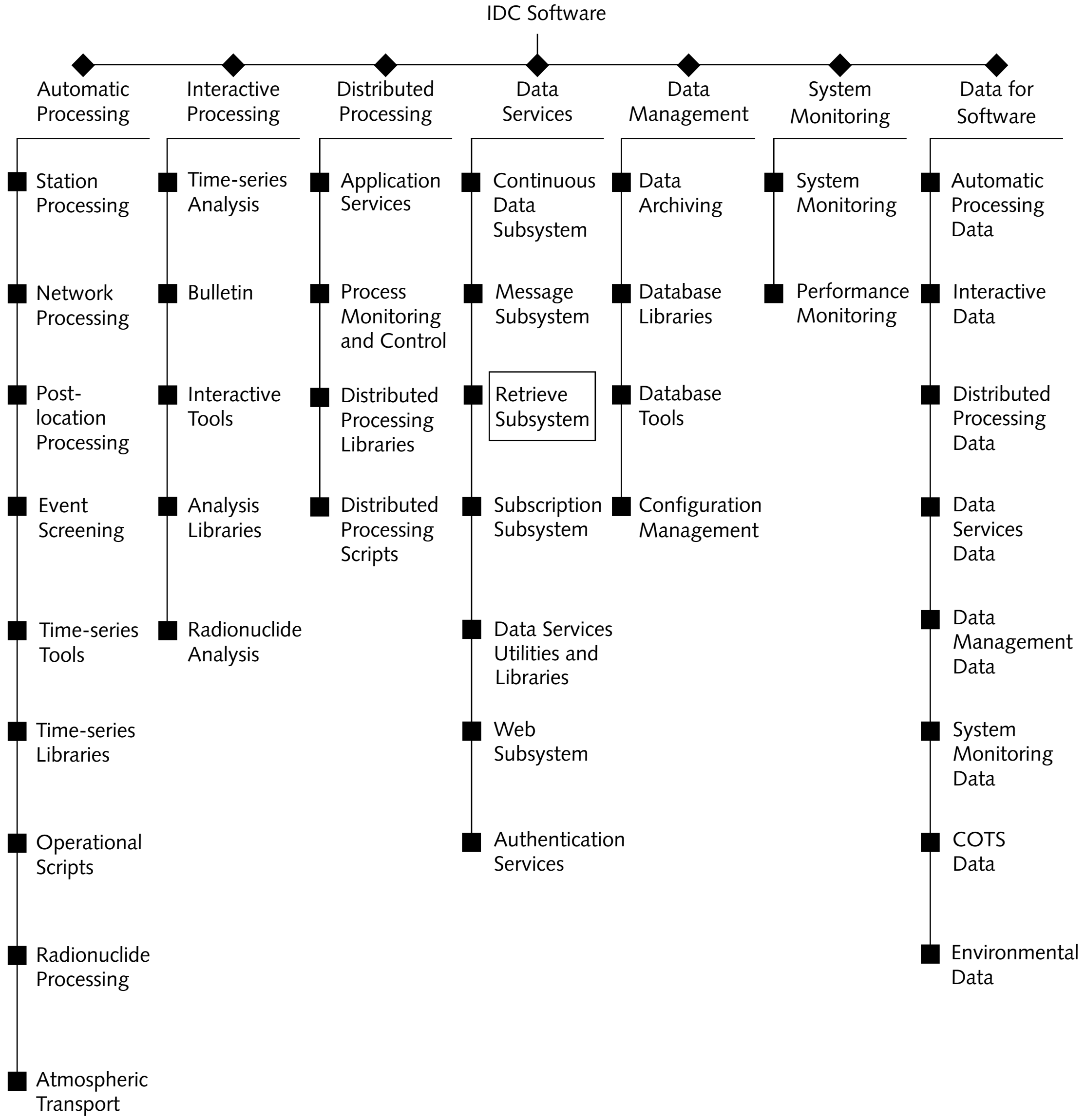


FIGURE 1. IDC SOFTWARE CONFIGURATION HIERARCHY

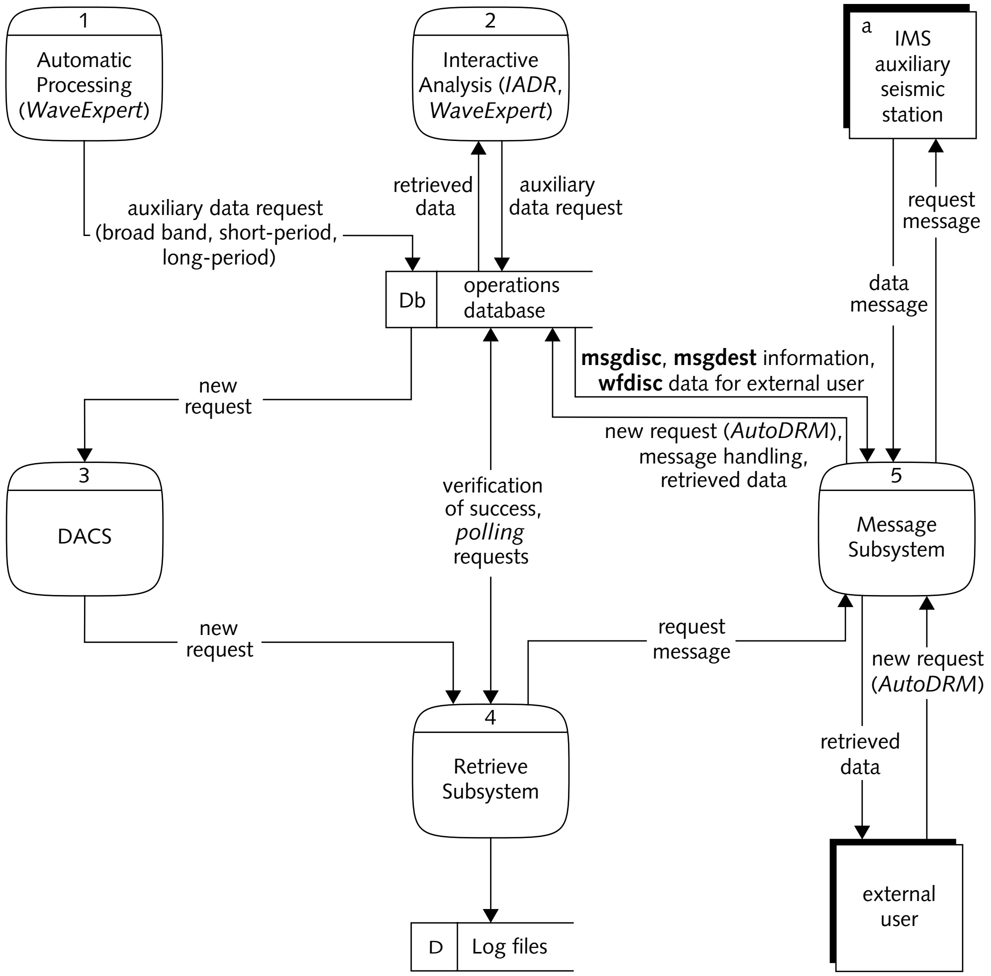

1
Automatic Processing (*WaveExpert*)
2
Interactive Analysis (*IADR*, *WaveExpert*)
a
IMS auxiliary seismic station
auxiliary data request (broad band, short-period, long-period)
retrieved data
auxiliary data request
request message
data message
Db
operations database
msgdisc, **msgdest** information, **wfdisc** data for external user
new request
new request (*AutoDRM*), message handling, retrieved data
3
DACS
verification of success, *polling* requests
5
Message Subsystem
new request
request message
new request (*AutoDRM*)
4
Retrieve Subsystem
retrieved data
D
Log files
external user

FIGURE 2. RELATIONSHIP OF THE RETRIEVE SUBSYSTEM TO OTHER SOFTWARE UNITS OF THE DATA SERVICES CSCI

STATUS OF DEVELOPMENT

The Retrieve Subsystem development is complete. The IDC Release 3 version of the Retrieve Subsystem includes new functionality to retrieve long-period data from IMS auxiliary stations through a retrieve pipeline. This functionality was added using configuration changes within the Distributed Application Control System (DACS).

FUNCTIONALITY

Overview

The Retrieve Subsystem provides the functionality to automatically retrieve seismic data from auxiliary IMS stations. These data are used by the Automatic Processing CSCI to refine results of Network Processing and Post-location Processing for identified events. Software from the Automatic Processing CSCI, the Distributed Processing CSCI, the Message Subsystem [IDC7.4.2], and the interactive auxiliary data request software (*IADR*) interface with the Retrieve Subsystem to perform auxiliary data retrieval.

Software components (for example, *WaveExpert*) generate requests for auxiliary data by adding rows to the **request** table. The Retrieve Subsystem translates these requests into IMS 1.0 or GSE 2.0 format data requests and passes them to the Message Subsystem, which sends out requests and parses data responses. The Retrieve Subsystem then reconciles the response with the request and retries the request if no response has been received.

Origins of Requests

Auxiliary data requests are provided to the Retrieve Subsystem from several sources. First, the DACS uses *WaveExpert* to request specific auxiliary data for the SEL1 and SEL2 pipelines and also all available auxiliary station long-period data. Second, the Retrieve Subsystem itself, through the program *polling*, creates daily requests for all IMS auxiliary stations to generate statistics on station availability. Third, *IADR* allows analysts to request auxiliary data. *IADR* uses *WaveExpert* to assess and create requests. Finally, *AutoDRM* generates auxiliary data requests when it receives requests for auxiliary data not present at the data center. All of these components submit requests into the Retrieve Subsystem through the **request** table.

WaveExpert submits information about a request for auxiliary data into the **request** table based on its own calculations. *WaveExpert* verifies that the desired auxiliary data intervals are not currently available in the **wfdisc** table and that the data have not already been requested before writing the new request to the **request** table.

The Perl script *polling* is executed daily from *cron*. The **sitepoll** table is read by *polling* for auxiliary stations and channels to use, and *polling* creates one-minute-long interval requests for them.

Request Process Flow

Figure 3 shows the Retrieve Subsystem processing flow. Request processing begins when a new auxiliary data request record is added to the **request** table. The DACS uses *WaveGet_server* to periodically poll the **request** table. When *WaveGet_server* encounters a new request in the **request** table, it passes a DACS message containing information about requested data to *tuxshell.* The *tuxshell* then spawns *dispatch* as a child. The *dispatch* program adds the auxiliary station's AutoDRM address provided in `dispatch.par` and converts the DACS message into an IMS 1.0 or GSE 2.0 format request message.

The *dispatch* program invokes *MessageSend*, which adds message header and footer lines and writes the request messages to a permanent storage directory (configured in `MessageSend.par`). *MessageSend* also inserts the information about the stored message in the **msgdisc** and **msgdest** database tables with **msgdest**.*status*=`PENDING`. These tables serve as the interface to the Message Subsystem.

The Message Subsystem distributes the data request messages via UNIX mail to the IMS auxiliary stations. Responses from auxiliary stations are parsed by the Message Subsystem, received data is stored in the filesystem, and information describing received data intervals is inserted into the **wfdisc** table.

The Retrieve Subsystem verifies auxiliary data retrieval by comparing the data requested to data available in the database. *WaveAlert* periodically compares information in the **wfdisc** and **outage** tables to the **request** table. *WaveAlert* updates the *state* of requests to `done-success` if requested data have been successfully received. *WaveAlert* updates the *state* of requests to `done-partial` if some of the requested data have been received. *WaveAlert* updates the *state* of requests to `retry` if no data have been received after a configurable time interval and then requests are resubmitted to auxiliary stations. *WaveAlert* updates the *state* of requests to `done-no-data` after another configurable time interval if the resubmitted request attempts are unsuccessful.

When requests are in the states `done-success`, `done-partial` or `done-no-data`, no further processing is performed by the Retrieve Subsystem. Unsuccessful requests are re-requested after *cleanup-time* and set to `done-no-data` after *max-submit-time*. Requests are set to `dispatch-failed` when *dispatch* is unable to create a request message for the **request** record after a configurable number of attempts.

Features and Capabilities

The Retrieve Subsystem handles an increased number of requests during extreme activity. Operators are able to start additional *WaveAlert* processes to facilitate request processing. For more information, refer to "Advanced Procedures" on page 17.

The Retrieve Subsystem accommodates additional auxiliary stations joining the IMS network in the future. Operators can add a new station's configuration information to the parameter file for *dispatch*, add an entry in the **sitepoll** database table, and add the station name to *GSE-list* in `shared.par`. This is all that is required for the Retrieve Subsystem to recognize new stations.

The Retrieve Subsystem uses other units in the IDC software for efficiency. This includes message handling by the Message Subsystem and DACS control of processes. The Retrieve Subsystem interfaces are minimal as the subsystem communicates with other software through the ORACLE database (*libgdi*), uses standard configuration options (*libpar*), and uses standard logging libraries (*liblogout*).

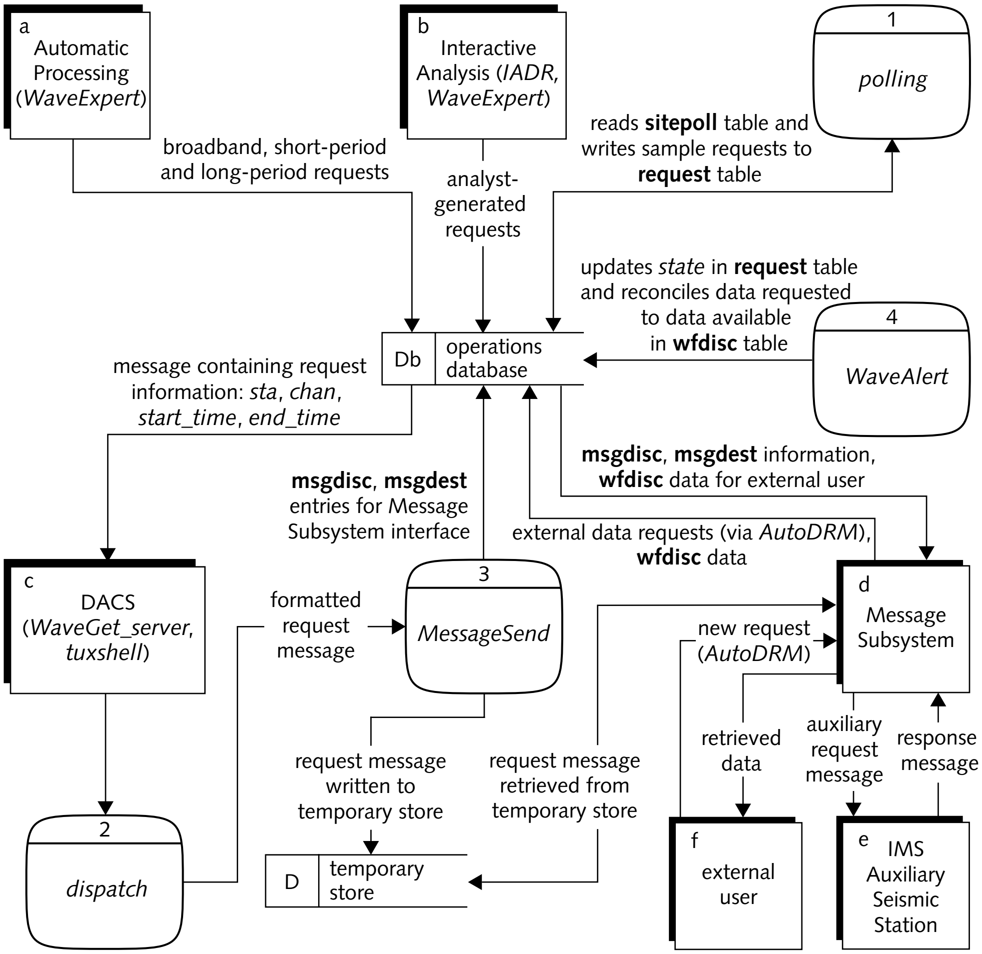

a
Automatic Processing (WaveExpert)
b
Interactive Analysis (IADR, WaveExpert)
1
polling
broadband, short-period and long-period requests
analyst-generated requests
reads sitepoll table and writes sample requests to request table
updates state in request table and reconciles data requested to data available in wfdisc table
4
WaveAlert
Db
operations database
message containing request information: sta, chan, start_time, end_time
msgdisc, msgdest entries for Message Subsystem interface
msgdisc, msgdest information, wfdisc data for external user
external data requests (via AutoDRM), wfdisc data
c
DACS (WaveGet_server, tuxshell)
formatted request message
3
MessageSend
d
Message Subsystem
new request (AutoDRM)
retrieved data
auxiliary request message
response message
request message written to temporary store
request message retrieved from temporary store
2
dispatch
D
temporary store
f
external user
e
IMS Auxiliary Seismic Station

FIGURE 3. RETRIEVE SUBSYSTEM INTERFACES AND PROCESS FLOW

Performance Characteristics

Retrieve Subsystem performance for a week at the Prototype International Data Centre (PIDC) is used as an example in Table 1. The four days in April have requests processing in states `requested` and `running`. The days in March are beyond the *max-submit-time* threshold and *WaveAlert* updates any requests still attempting to retrieve data from auxiliary stations after this threshold to `done-no-data`. On average, PIDC software systems sent 2,813 requests for auxiliary data for these days, and on average, 2,594 responses or 92 percent were successfully received with data from auxiliary stations. In the time interval shown in Table 1, which represents typical Retrieve Subsystem performance, a request is processed from *state* `requested` to `done-success` in an average of five minutes.

TABLE 1: PERFORMANCE STATISTICS, 29 MARCH THROUGH 4 APRIL 2001

		State						
Date	**Total Number of Requests**	**requested**	**running**	**retry**	**done-success**	**done-partial**	**done-no-data**	**failed**
4 April 2001	3059	3	129	0	2927	0	0	0
3 April 2001	2333	0	171	0	2161	1	0	0
2 April 2001	2491	0	192	0	2287	0	12	0
1 April 2001	3959	0	65	0	3612	0	6	276
31 March 2001	3270	0	0	0	3040	0	3	227
30 March 2001	2277	0	0	0	2058	0	0	219
29 March 2001	2301	0	0	0	2070	0	0	231

Related Tools

The *WorkFlow* and *RequestFlow* graphical user interface (GUI) tools are helpful for monitoring the operation of the Retrieve Subsystem. These tools monitor the database and track interval processing. Note that *RequestFlow* is actually the *WorkFlow* executable used with a special parameter file. *WorkFlow* displays interval processing status. *RequestFlow* displays the status of the *state* field in the **request** table.

INVENTORY

The Retrieve Subsystem relies on the functionality of DACS and the Message Subsystem to operate. These Subsystems must be installed before the Retrieve Subsystem can function. Components directly used by the Retrieve Subsystem are listed in Table 2.

TABLE 2: RETRIEVE SUBSYSTEM INVENTORY

Item	Type
dispatch	Executable
MessageSend	Executable
polling	Executable
WaveAlert	Executable
`WaveAlert.par`	Parameter file
`MessageSend.par`	Parameter file
`dispatch.par`	Parameter file
libconvert	Common Library
libgdi	Common Library
libgeog	Common Library
libgsemsg	Data Services Library
libgsewf	Data Services Library
liblogout	Common Library

TABLE 2: RETRIEVE SUBSYSTEM INVENTORY (CONTINUED)

Item	Type
libmisc	Common Library
libpar	Common Library
libstdtime	Common Library
libtable	Common Library
libwfm	Common Library
libdl	COTS Library
libf77	COTS Library
libm	COTS Library
libnsl	COTS Library
libsocket	COTS Library
lastid	System Database table (read/write)
msgdest	Message Subsystem database table (write)
msgdisc	Message Subsystem database table (write)
outage	Message Subsystem database table (read)
request	Retrieve Subsystem database table (read/write)
sitepoll	Retrieve Subsystem database table (read)
wfdisc	System database table (read)

ENVIRONMENT AND STATES OF OPERATION

The following paragraphs describe the hardware and commercial-off-the-shelf (COTS) software required to operate the Retrieve Subsystem.

Software Environment

The Retrieve Subsystem software is designed for Solaris 7 and ORACLE 8i. The Retrieve Subsystem is controlled by the DACS, which requires Tuxedo 6.5.

Hardware Environment

The Retrieve Subsystem is designed to run on a UNIX workstation such as the SPARCstation 20/612. The Retrieve Subsystem has minimal disk space requirements and only needs enough space for storing par files and executables. Typically, the hardware is configured with 64 MB of memory. The Retrieve Subsystem must obtain other services (such as database access and Inter-Process Communications resources) over its network connection to other computers. Figure 4 shows a representative hardware configuration.

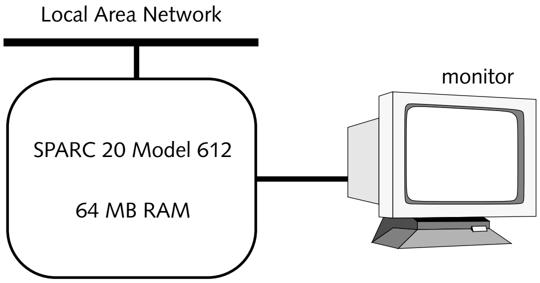


FIGURE 4. REPRESENTATIVE HARDWARE CONFIGURATION FOR THE RETRIEVE SUBSYSTEM

Normal Operational State

The Retrieve Subsystem is designed to run automatically without user assistance. *WaveAlert* is the only process that should run continuously. The remainder of the Retrieve Subsystem programs are either called as child processes or by *cron*. The DACS controls *dispatch* through *WaveGet_server* and *tuxshell*. These applications call *dispatch* and provide input from the **request** table.

Contingencies/Alternate States of Operation

The Retrieve Subsystem is generally run in its normal operational state. A limited user interface is available with the Retrieve Subsystem and these operations are explained in "Advanced Procedures" on page 17.

Chapter 2: Operational Procedures

This chapter provides instructions for using the Retrieve Subsystem software and includes the following topics:

- Software Startup
- Software Shutdown
- Basic Procedures
- Advanced Procedures
- Maintenance
- Security

Chapter 2: Operational Procedures

SOFTWARE STARTUP

The Retrieve Subsystem is designed to run automatically without user assistance. *dispatch* is invoked as a child of *tuxshell*, *MessageSend* is invoked as a child of *dispatch*, and *polling* is invoked as a child of *cron*. *WaveAlert* is the only Retrieve Subsystem program to run continuously and is started with Message Subsystem programs using the script *keep_msg_alive*.

The DACS controls *dispatch*. DACS calls *WaveGet_server* and *tuxshell*, which calls *dispatch* and provides request information.

SOFTWARE SHUTDOWN

The Retrieve Subsystem components *dispatch*, *MessageSend*, and *polling* terminate automatically when they have completed processing. *WaveAlert* runs continuously and is shut down by killing the UNIX process. The *keep_msg_alive* script must be disabled to prevent *WaveAlert* from being restarted. To stop the Retrieve Subsystem from creating new request messages, the corresponding DACS processes (*tuxshell* and *WaveGet*) must be shut down or stalled (see [IDC6.5.2Rev0.1]).

BASIC PROCEDURES

The Retrieve Subsystem runs automatically and almost no effort is required for daily operation of the subsystem. Users should regularly check the status of the Retrieve Subsystem. This is discussed in "Monitoring" on page 24.

Obtaining Help

Online help is available for the Retrieve Subsystem as UNIX man pages for the Retrieve Subsystem components *dispatch*, *MessageSend*, and *WaveAlert*. For information on diagnosing Retrieve Subsystem problems, see "Chapter 3: Troubleshooting" on page 23.

ADVANCED PROCEDURES

The Retrieve Subsystem interfaces with other subsystems through database table records. These records can be modified and/or new records can be created in special circumstances to alter processing.

Modify the ORACLE database records to process, reprocess, or to halt processing of intervals for auxiliary data requests. Auxiliary data processing is controlled by updating the *state* field in the **request** table to the desired value. Table 3 shows the *state* values used in the **request** table.

TABLE 3: STATES OF REQUESTS IN THE REQUEST TABLE

State	Definition	Author
`requested`	Initial *state* for new requests, waiting to be picked up by *WaveGet*.	all requestors (for example: *WaveExpert*, *polling*, and so on)
`queued`	*WaveGet_server* has picked the request up and sent an IPC message to the DACS, request is in the dispatch-queue.	*WaveGet_server*
`running`	*dispatch* and *MessageSend* have processed the request and passed it on to the Message Subsystem, which sends it to the IMS station. The Retrieve Subsystem is waiting for a response from the station.	*tuxshell-dispatch*

TABLE 3: STATES OF REQUESTS IN THE REQUEST TABLE (CONTINUED)

State	Definition	Author
`dispatch-failed`	*dispatch* failed to format the request, or *WaveGet_server* updated the request to *state* `dispatch-failed` after having retried the request a *max-retry* number of times unsuccessfully.	*WaveGet_server*, *tuxshell-dispatch*
`done-success`	Data have been successfully retrieved.	*WaveAlert*
`done-partial`	Data have been partially retrieved after *cleanup-time* and the request will not be retried.	*WaveAlert*
`retry`	Data have not been retrieved after *cleanup-time* and request is not older than *max-submit-time*, so it is set to *state* `retry` by *WaveAlert* and will be picked up again by *WaveGet_server*.	*WaveAlert*
`dispatch-retry`	If *dispatch* fails to format the request, *tuxshell* will update its *state* to `dispatch-retry` and call *dispatch* again before it finally sets the *state* to `dispatch-failed`.	*tuxshell-dispatch*
`done-no-data`	Data have not been retrieved after *max-submit-time*.	*WaveAlert*

Use SQL*Plus to log into the database and execute SQL*Plus commands to change an interval's processing status.

Request specific auxiliary station data by creating a new record in the **request** table. Insert a row in the **request** table with values for the *start_time*, *end_time*, *sta*, and other fields with the *state* set to `requested` (refer to "Database" on page 44 for a description of the **request** table fields). The Retrieve Subsystem processes the request after the record is created with *state*=`requested`.

Additionally, multiple instances of *WaveAlert* can be invoked during heavy Retrieve Subsystem activity. Invoking multiple processes may be desired as the number of auxiliary stations in the IMS network increases. Multiple *WaveAlert* instances can increase the efficiency of the subsystem by proportionately reducing the time used to verify requested auxiliary data that have been parsed into the IDC database (in other words, running two *WaveAlert* instances would process in half the time, running three *WaveAlert* instances would process in a third of the time). For more information on manually invoking a *WaveAlert* process, refer to "Initiating Operations" on page 50.

MAINTENANCE

Operator maintenance duties are minimal for the Retrieve Subsystem. The subsystem generates log files, database entries, and message files that are either automatically maintained or maintained by operators of other subsystems.

Log files for the Retrieve Subsystem recycle after a parameter-specified number of instances. Log file recycling is explained in "Chapter 3: Troubleshooting" on page 23. Database table size is controlled through data migration and database table purges. This maintenance is performed by a separate subsystem within the Data Management CSCI, and no maintenance is required by Retrieve Subsystem operators. Request messages are stored by the Message Subsystem, and these files are maintained by Message Subsystem operators.

As new auxiliary stations are brought online, update the **sitepoll** table with the relevant information. This enables *polling* to perform its daily query on the new station. Other Retrieve Subsystem programs receive station information through *WaveGet_server* and its use of the **affiliation** table. Add the new station to *GSE-list* in `shared.par` (for *WaveGet*) and add the station's *AutoDRM* address in `dispatch.par`. No other station maintenance is required.

Regularly check the progress of the Retrieve Subsystem. Procedures for this are discussed in "Monitoring" on page 24.

SECURITY

Update permissions for the database accounts and configuration data files should be protected to prevent the potential removal, addition, or manipulation of information in the account. For example, data from a particular station can be blocked from processing by simply removing the station from the *GSE-list* or `dispatch.par` or manually updating the *state* of a request from `requested` to a final processing state such as `done-success`. However, this ability can also be used to reprocess requests when recovering from major system failures by updating the *state* field to `requested`.

Additionally, inbound and outbound messages stored as files can be removed from the UNIX system by operators with the appropriate UNIX file permissions. A possible modification of content within message files will be detected if the messages are digitally signed, as this would allow the authentication software to detect tampered files.

Passwords

Passwords are not written to Retrieve Subsystem log files. Database passwords and accounts are stored in `process.par`. Anyone with permission to view this file can retrieve database passwords and manipulate database accounts. Access to these files is controlled by UNIX permissions, and the operations manager controls the UNIX group membership required to access `process.par`.

Marking/Storing Controlled Outputs

All incoming and outgoing messages are stored as files in the directory structure. Access to these files is controlled by UNIX permissions.

Email is used as the primary method of transmitting messages and content of the messages may potentially be intercepted during transmission. Authentication is supported with IDC software. Messages can only be digitally signed with properly issued public and private keys and when authentication is enabled. Any tampered or otherwise corrupted messages are detected by the relevant subsystem via the capabilities of the Authentication Services component of the Data Services CSCI. The Message Subsystem can be configured to ignore any message that fails signature validation.

Chapter 3: Troubleshooting

This chapter describes how to identify and correct problems related to the Retrieve Subsystem and includes the following topics:

- Monitoring
- Interpreting Error Messages
- Solving Common Problems
- Reporting Problems

Chapter 3: Troubleshooting

MONITORING

Several methods can be used to verify that the Retrieve Subsystem programs are functioning properly. The primary method of monitoring the Retrieve Subsystem is through the use of the *RequestFlow* GUI tool. Other methods of monitoring the subsystem include tracking requests in the **request** database table using SQL*Plus commands, screening log files for problems, and verifying recent process activity. These methods are described in the following sections.

RequestFlow

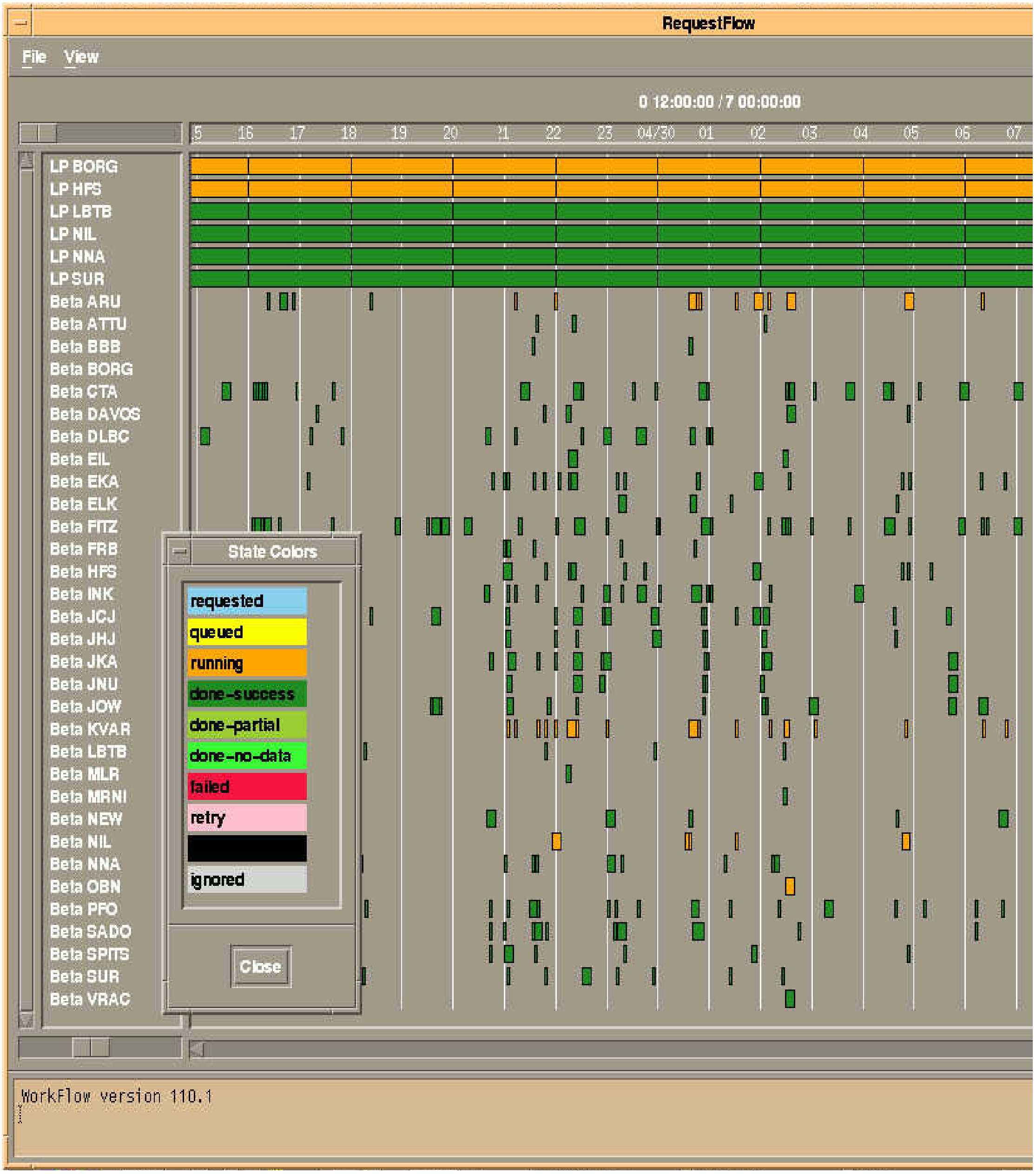
The recommend method for monitoring the Retrieve Subsystem is through the use of the *RequestFlow* GUI. The *RequestFlow* program graphically displays the status of requests in the **request** table. Intervals are depicted in different colors to represent the status of processing. Start the *RequestFlow* GUI by executing the *WorkFlow* binary with the `RequestFlow.par` parameter file. An example of the command line execution of *RequestFlow* follows (the `-name` option creates a title for the window bar):

```
%setenv DISPLAY pickle:0.0
%/home/cmss/rel/bin/WorkFlow \
par=/cmss/config/app_config/distributed/WorkFlow/\
RequestFlow.par -name RequestFlow
```

Alternatively, use the *start* script to start *RequestFlow*:

```
%setenv DISPLAY_RequestFlow pickle:0.0
%start program=RequestFlow
```

When *RequestFlow* is started, a new window opens on the user's screen (Figure 5).



```
WorkFlow version 110.1
```

FIGURE 5. REQUESTFLOW GUI

All auxiliary stations are displayed in the left column in the *RequestFlow* GUI window. Auxiliary stations providing long-period data and auxiliary stations providing data for redirected user requests are displayed above regular auxiliary stations. Time periods are labeled across the top of the window. Requested intervals are shown as colored blocks on the screen. As auxiliary data are retrieved, the interval blocks change colors. Clicking on an interval block with a mouse reveals information about the request including the station, time interval requested, the request's status, and the request ID.

Monitoring the request Table

Monitoring the **request** table is similar to monitoring with *RequestFlow* as both of these monitoring activities examine **request**.*status*. In this method, operators use SQL*Plus commands to query for **request** table records. *RequestFlow* is useful for daily monitoring and general evaluation of auxiliary data processing, while this method is useful for examining specific requests or obtaining statistical summaries when verifying the subsystem configuration.

As requests are processed by the Retrieve Subsystem, the record's *state* in the **request** table is progressively updated through the states listed in Table 3 on page 17.

To check the processing of the Retrieve Subsystem, log into the database and select a new record with, for example, *state* `running`. Use the unique *reqid* value to query the selected record and after a moment, repeat the same query to ensure the record is being processed (in other words, updated to another *state*). The *state* `running` means that the subsystem is waiting for a response from the IMS station, so effective processing at this stage depends on external factors. For example:

```
SQL> select reqid, sta, chan, state from idcx.request
  2  where state='running';

     REQID STA    CHAN     STATE
---------- ------ -------- ----------------
     34415 PFO    bz       running
     34416 PFO    bn       running
     34417 PFO    be       running
```

```
SQL> select sta, chan, state from idcx.request
  2  where reqid like '34415';

STA    CHAN     STATE
------ -------- ----------------
PFO    bz       running


SQL> select sta, chan, state from idcx.request
  2  where reqid like '34415';

STA    CHAN     STATE
------ -------- ----------------
PFO    bz       done-success
```

Operators can use the following SQL query to find the most recent auxiliary data successfully retrieved:

```
SQL> select max(start_time) from idcx.request
  2  where state='done-success';


MAX(START_TIME)
---------------
      982915200


% e2h 982915200
  982915200.00000 2001054  2001/02/23 08:00:00.00000 Feb Fri
```

Screening Log Files

The log files of individual programs can be screened for problems. Log files for *MessageSend* and *WaveAlert* are written to the same directory as the Message Subsystem program log files (`/logs/msg/`). Log files for *dispatch* and the *tuxshell* parent process are written to the general log directory (`/logs`) under individual Julian date directories. The Perl script *polling* does not write a log file. After moving to the log file directory, use the UNIX `ls -l` command to list log files for the programs.

```
% cd /logs/msg
% ls -l MessageSen*
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:24 MessageSend
```

```
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:24 MessageSend.1
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.10
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.11
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.12
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.13
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.14
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.15
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.16
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:15 MessageSend.17
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:15 MessageSend.18
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:13 MessageSend.19
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:23 MessageSend.2
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:13 MessageSend.20
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:23 MessageSend.3
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:23 MessageSend.4
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.5
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.6
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.7
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.8
-rw-rw-rw-   1 auto     cmss        195 Feb 22 11:18 MessageSend.9
```

The list above displays log files available for *MessageSend*. The current log file is the program's name without a numerical suffix. To view the log file for the previous instance of the program, examine *progname*`.1`. When the program is executed again, *progname*`.1` becomes *progname*`.2` and this renumbering cycle continues until log files recycle.

Use the UNIX `grep` command to search for problems:

```
% grep fatal MessageSen*
% grep error MessageSen*
% grep warning MessageSen*
```

To monitor the operation of *MessageSend* or *WaveAlert* in real time, use the UNIX `tail -f` command to monitor lines being written to the log file.

```
% tail -f WaveAlert
WaveAlert[gap]: in outage table :  0/180.0 seconds (0.0%): running
WaveAlert[gap]: both tables : wfdisc + outage : 0/180.0 seconds
                (0.0%): running
WaveAlert[gap]: checking SUR  sz:  982464184.7 (180)
```

```
WaveAlert[gap]: in wfdisc table ;  0/180.0 seconds (0.0%): running
WaveAlert[gap]: in outage table :  0/180.0 seconds (0.0%): running
WaveAlert[gap]: both tables : wfdisc + outage : 0/180.0 seconds
                (0.0%) running
```

Verifying Active Processes

Recent program activity can be checked by looking in current log files for `dispatch`, `MessageSend`, and `WaveAlert`. Additionally, verify that a *WaveAlert* process is running.

Use the UNIX `ls -la` command to display log file dates. Older log files indicate that a program is not currently active (either it is "hung" or it has exited and not restarted). Examine old log files for any indication of errors as described in "Screening Log Files" on page 27.

```
% cd /logs/msg
% ls -l WaveAlert
-rw-rw-r--   1 auto     cmss     2878925 Feb 22 12:08 WaveAlert

% ls -l MessageSend
-rw-rw-rw-   1 auto     cmss         195 Feb 22 11:24 MessageSend

% cd /logs/2001054
% ls -l dispatch
drwxrwxr-x   2 auto     cmss         512 Feb 21 20:37 dispatch
```

WaveAlert is the only Retrieve Subsystem program that runs continuously in a loop cycle. If records in the **request** table are not being updated (to `done-success`, `done-partial`, `done-no-data`, or `retry`), *WaveAlert* may be "hung" or have exited. Log onto the machine hosting the Retrieve Subsystem and use the UNIX `ps -ef | grep` command to find an active *WaveAlert* process as follows:

```
% ps -ef | grep WaveAlert
auto  2023 1 0 Feb 06 24:36 WaveAlert
      par=/cmss/config/app_config/messages/WaveAlert/WaveAlert.par
```

WaveAlert should be restarted if there is not an active process. *WaveAlert* is started from a command line with a parameter file. Refer to "Initiating Operations" on page 50 for details. Examine the most recent log file for *WaveAlert,* as described in "Screening Log Files" on page 27 for troubleshooting information.

Kill and restart *WaveAlert* if the program appears to be "hung." This condition is indicated by the presence of a *WaveAlert* process, but old log files and stagnant states in the **request** table (for example not being updated to final states). Use the UNIX `kill` command to terminate the process and then restart *WaveAlert* as described in "Initiating Operations" on page 50.

```
% ps -ef | grep WaveAlert

auto  2023 1 0 Feb 06 24:36 WaveAlert
      par=/cmss/config/app_config/messages/WaveAlert/WaveAlert.par

% kill -9 2023
```

INTERPRETING ERROR MESSAGES

This section describes common error messages that the Retrieve Subsystem writes to log files. Only *dispatch*, *MessageSend*, and *WaveAlert* write errors to log files.

dispatch

Message: `Undefined: …`

Description: Upon execution, required parameters were not defined for *dispatch*.

Action: Add or set the required parameters in the *dispatch* parameter file.

Message: `Cannot start process: ...`

Description: Upon execution, *dispatch* was unable to open a pipe to *MessageSend* to start as a child process.

Action: Verify that the *MessageSend* binary is properly installed and that the *GSE-command* parameter in `dispatch.par` contains the proper command.

MessageSend

Message: `fatal ... cannot open database ...`

Description: *MessageSend* is unable to access the database specified in the parameter file.

Action: Verify the accuracy of the database account and password provided to *MessageSend* in its parameter file. Verify that a database license for connection is available.

WaveAlert

Message: `fatal ... cannot open database ...`

Description: *WaveAlert* is unable to access the database specified in the parameter file.

Action: Verify the accuracy of the database account and password provided to *WaveAlert* in its parameter file.

Message: `fatal … required to run WaveAlert`

Description: Upon execution, *WaveAlert* was not provided a required parameter in its par file.

Action: Add the required parameter to the *WaveAlert* parameter file.

Message: `fatal – can’t get next otgid from lastid`

Description: *WaveAlert* is unable to read the outage ID value from the **lastid** database table.

Action: Verify that a value exists for *otgid* in the **lastid** table. If necessary, initialize the *otgid* value with a non-negative number.

SOLVING COMMON PROBLEMS

The Retrieve Subsystem relies heavily on the proper operation of other software subsystems. The Retrieve Subsystem itself runs automatically and is not known to have any common problems. The behavior of other subsystems affects Retrieve Subsystem performance. Problems with other software on which the Retrieve Subsystem depends can halt auxiliary data processing.

Processes such as message mailing and the storing of auxiliary data depend on mail servers and Message Subsystem processing. Other possible disruptions in the Retrieve Subsystem operation could be attributed to problems with the *keep_msg_alive* script for *WaveAlert*, proper DACS functionality, and proper operation of *WaveGet_server*. Problems with requestor processes that create new entries in the **request** table can lead to bad requests that cannot be processed by the Retrieve Subsystem. Additionally, because these subsystems all interface through the ORACLE database, ORACLE standard database errors can be encountered as well (for example, database connection errors, and update errors).

Error Recovery

Little risk is associated with a temporary shutdown of the Retrieve Subsystem due to either a failure of one of its programs or a failure associated with another subsystem on which it relies. If Retrieve Subsystem processing is interrupted, it will process requests based on the *state* in the **request** table when it resumes operation. For example, if a mail server crashes, the Message Subsystem is shut down, or the database is temporarily unavailable, the Retrieve Subsystem will simply retroactively process a backlog of requested auxiliary data requests when any problems have been corrected.

REPORTING PROBLEMS

The following procedures are recommended for reporting problems with the application software:

1. Diagnose the problem as far as possible.
2. Record information regarding symptoms and conditions at the time of the software failure.
3. Retain copies of relevant sections of application log files.
4. Contact the provider or maintainer of the software for problem resolution if local changes of the environment or configuration are not sufficient.

Chapter 4: Installation Procedures

This chapter provides instructions for installing the software and includes the following topics:

- Preparation
- Executable Files
- Configuration Data Files
- Database
- Tuxedo Files
- Initiating Operations
- Validating Installation

Chapter 4: Installation Procedures

PREPARATION

Select a host machine for the Retrieve Subsystem and have binaries and parameter files installed. Additionally, the Message Subsystem and DACS must be installed and operational prior to startup of the Retrieve Subsystem to facilitate message distribution and data parsing. Parameter files provided with the Retrieve Subsystem programs must be edited for site specific paths. Details on parameter file editing are provided later in this chapter.

Use the *RequestFlow* GUI to monitor the Retrieve Subsystem. Select a machine to host and execute *RequestFlow*. This program is typically executed and displayed on the same machine hosting the *WorkFlow* GUI. *RequestFlow* is a derivation of the *WorkFlow* executable using the `RequestFlow.par` parameter file. `RequestFlow.par` should be installed in the same directory as `WorkFlow.par`.

Obtaining Released Software

The software is obtained via FTP from a remote site or via a physical medium, such as tape or CD-ROM. The software and associated configuration data files are stored as one or more tar files. The software and data files are first transferred via FTP or copied from the physical medium to an appropriate location on a local hard disk. The tar files are then untarred into a standard UNIX directory structure.

Hardware Mapping

Select the hardware on which to run the software components. Software components are generally mapped to hardware to be roughly consistent with the software configuration model.

UNIX System

The Retrieve Subsystem uses the UNIX filesystem to store parameter and log files for *dispatch*, *MessageSend*, and *WaveAlert*. The Retrieve Subsystem indirectly requires the use of a mailer through the Message Subsystem. Configuration of UNIX for a mailer is described in the Installation Procedures section of the *Message Subsystem Software User Manual* [IDC6.5.19].

Firewall

The Retrieve Subsystem interfaces directly with other IDC software systems and does not require any firewall configuration. The Retrieve Subsystem relies on the Message Subsystem for all external communication. Refer to the Installation Procedures section of [IDC6.5.19] for a description of the Message Subsystem firewall configuration.

EXECUTABLE FILES

The program *dispatch* should be installed in the `/cmss/rel/bin` directory and the script *polling* should be installed in the `/cmss/scripts` directory. Both *dispatch* and *polling* should be accessible by the Retrieve Subsystem host machine. The executable files *MessageSend* and *WaveAlert* should be installed in the `/cmss/rel/bin` directory and be accessible by the Message Subsystem host machine.

CONFIGURATION DATA FILES

The parameter files provided with the software include default values for all parameters. For efficiency, Retrieve Subsystem par files use global variables for path names where possible. Parameter files for *dispatch*, *MessageAlert*, and *MessageSend* include paths that must be edited for the IDC environment.

The global parameter files `process.par`, `shared.par`, and `global.shenv` also contain site-specific path names used by the Retrieve Subsystem that must be modified for the IDC environment. These files contain global variables for all operational software and only some parameters need modification. The `process.par` and shared.par files are located in the directory /cmss/config/system_specs. The file global.shenv is located in the subdirectory /cmss/config/system_specs/env. The UNIX account from which the Retrieve Subsystem is run must be configured to read the file process.par at login. This is typically done in the .cshrc file in the user's home directory by reading the global.env file, which in turn reads process.par if the user's *CMS_MODE* environment variable is set to process.

Parameters for each of these par files that are pertinent to the Retrieve Subsystem are listed in the examples below. Italicized portions of the parameters must be replaced with values appropriate for the IDC environment.

process.par

```
IDCXDB=account/password@machine
par=$(CMS_CONFIG)/system_specs/shared.par
```

shared.par

```
timezone-difference=0.208333333 # 5 hours (U.S. Eastern Standard Time)
#timezone-difference=0.166666 # 4 hours (U.S. Eastern Daylight Time)
operator=pipeline
AUTO=pipeline
CMSS=nmrd
domain=cmr.gov
# data-center is the name of the data center
data-center=PIDC
# These parameters are truly shared and may be derived to some
# degree from the site specific settings
#
DATABASE_VENDOR=oracle
max-wfdisc-duration=14500
extension_time=$(max-wfdisc-duration)
LOGDIR=/logs
AUXDIR=/aux
```

```
TUXBASE_ANALYSIS=/var/tuxedo/PIDC70_analysis
QPATH=/var/tuxedo/PIDC70_process
QPATH_ANALYSIS=/var/tuxedo/PIDC70_analysis
RELDIR=$(CMS_HOME)
RELBIN=$(RELDIR)/bin
SQLDIR=$(CMS_CONFIG)/system_specs/sql
GLOBALDIR=$(CMS_CONFIG)/system_specs
SCRIPTSBIN=$(CMS_SCRIPTS)/bin
# Station Lists
PRI_LIST="’ABKT’,’ARCES’,…,’YKA’,’ZAL’"
HYD_LIST="’ASC23’,’ASC24’,…,’WK30’,’WK31’"
INF_LIST="’LSAR’,’NVIAR’,…,’IS10’,’IS59’"
DP_LIST="$(PRI_LIST),$(HYD_LIST),$(INF_LIST)"
# Auxiliary Station Lists
GSE-list= "ALQ,ARU,ATTU,…,SUR,TKL,VRAC"
DISTRIBUTED-DIR=$(CMS_CONFIG)/app_config/distributed
INTERACTIVE-DIR=$(CMS_CONFIG)/app_config/interactive
MESSAGES-DIR=$(CMS_CONFIG)/app_config/messages
MISCSPECS-DIR=$(CMS_CONFIG)/app_config/misc
STATION-DIR=$(CMS_CONFIG)/station_specs
# The following define the subsystem specific par files
#
DFXSPECS=$(CMS_CONFIG)/system_specs/DFX.par
AUTOMATIC=$(CMS_CONFIG)/system_specs/automatic.par
CONTINUOUS_DATA=$(CMS_CONFIG)/system_specs/cds.par
DISTRIBUTED=$(CMS_CONFIG)/system_specs/dacs.par
INTERACTIVE=$(CMS_CONFIG)/system_specs/interactive.par
MESSAGES=$(CMS_CONFIG)/system_specs/msgs.par
# RQHOST = Request Subsystem
RQHOST=TUXHOST3
# Backup Hosts
RQBAKHOST=TUXHOST2
TUXHOST2=niue
TUXHOST3=kuredu
```

global.shenv

```
APPDIR=/var/tuxedo/PIDC70_process/appdir:/cmss/rel/bin:/home/oracle/lib
CMS_APPS=/cmss/config/app_config
CMS_CONFIG=/cmss/config
CMS_HOME=/cmss/rel
CMS_MODE=process
CMS_RELEASE=PIDC70
CMS_SCRIPTS=/cmss/scripts
CONTRIB_HOME=/cmss/contrib
GDIHOME=/cmss/rel
GDI_HOME=/cmss/rel
GS_LIB=/cmss/local/lib/ghostscript/3.33:/cmss/local/lib/ghostscript/fonts
HOME=/home/nmrd
IMSPAR=/cmss/config/system_specs/process.par
LD_LIBRARY_PATH=/usr/dt/lib:/cmss/rel/lib:/home/oracle/lib:
    /cmss/cots/tuxedo/lib:/opt/SUNWspro/lib:/opt/S
LOCALHOME=/cmss/local
MANPATH=/usr/man:/usr/openwin/man:/usr/dt/man:/cmss/rel/doc/man:
    /cmss/local/man
ORACLE_HOME=/home/oracle
PATH=/usr/bin:/usr/sbin:/usr/dt/bin:/cmss/scripts/bin:/cmss/rel/bin:
    /cmss/contrib/bin:/home/oracle/bin:/cmss/cots/tuxedo/bin:/cmss/local/bin:
    /opt/SUNWspro/bin:/opt/SUNWmotif/bin:/usr/openwin/bin:
    /home/licensed/frame_5.5/bin:/opt/atria/bin:/opt/local/bin:
    /opt/local/adobe/Acrobat3/bin:/etc:/usr/ccs/bin:/usr/ucb:.
PERLLIB=/cmss/scripts/lib
PERLPATH=/cmss/local/bin/
```

WaveAlert.par

```
par=$(IMSPAR)
par=$(MESSAGES)
database=$(IDCXDB)
vendor=oracle
verbose=0
```

```
loop=1
sleep-time=60
cleanup-time=86400
lookback-time=345600
max-submit-time=345600
max-wfdisc-length=3600
complete-pct=95
complete-time=1
state-review-list="ignored queued requested running"
state-submit-list="running"

# WaveAlert will include a field "APPLICATION" in
# messages sent to AutoDRM. The value will be set
# with the value specified in the parameter
# application

application="WaveAlert_IDC"

log-directory=$(MSGLOGDIR)
log-name=WaveAlert
log-files=20
log-lines=50000
log-misc
log-gdi
log-gap
```

MessageSend.par

```
par=$(IMSPAR)
par=$(MESSAGES)
vendor=$(DATABASE_VENDOR)
database=$(EXPERTDB)
database=account/password:host:vendor
vendor=oracle
msgdir=$(MSGDIR)
address=operator email address
```

```
intidtype='reqid'

log-parse
log-database
log-directory=$(MSGLOGDIR)
log-name=MessageSend
log-files=20

support_smime=0
```

dispatch.par

```
par=$(IMSPAR)
par=$(MESSAGES)
vendor=$(DATABASE_VENDOR)
database=$(IDCXDB)
GSE-command="$(RELBIN)/MessageSend
        par=$(PARDIR)/MessageSend/MessageSend.par"
return-email-address=$(MSG_RETURN_ADDRESS)
ALQ-GSE-address=autodrm@gldfs.cr.usgs.gov
ARU-GSE-address=autodrm@fsuhub.gsras.ru
...
```

(Station and channel list in this par file - edited for length - operators will add stations and their channels to this file as maintenance duties.)

```
...
VRAC-GSE-address=autodrm@ipe.muni.cz
```

msgs.par

```
# %W% %G%
#

PARDIR=$(MESSAGES-DIR)

# Common log directory for all message system applications
```

```
MSGLOGDIR=$(LOGDIR)/msg

# Common message staging directory
MSG_TSD=$(MSGDIR)/msg_TSD
FTPMSG_TSD=$(MSGDIR)/auxftp_TSD

# message return address
MSG_RETURN_ADDRESS=messages@pidc.org

# Mail addresses to receive notifications from ParseData that
# a message failed for some reason.
MSG_OPERATOR="pipeline"

# path to aux data
# auxdir=$(AUXDIR)
dirname=$(MSG_TSD)

# Name of the RMS pipeline command:
RMSPIPELINE=/home/gardsops/rms_pipeline

# Parameters for authentication
support_smime=0
ca_cert_path=/data/ogma/msg/certs/cacerts
sign_cert_path=/data/ogma/msg/certs/usercerts/msgcert.pem
sign_key_path=/data/ogma/msg/certs/usercerts/msgreq.pem
pass_phrase=msg.phr
#
# Moved from other message applications
#
magtype_to_magname="mlppn ML ml ML mb_ave mb mb mb mb1 mb1 \
ms_ave Ms ms Ms ms1 Ms1"
magtype_to_magname_net="mlppn ML ml ML mb_ave mb mb mb \
mb1 mb1 mb_mle mbmle mb1mle mb1mx mb_ub mbub ms_ave Ms \
ms Ms ms1 Ms1 ms_mle msmle ms1mle ms1mx ms_ub msub"
magtype_to_magname_sta="mlppn ML ml ML mb_ave mb mb mb \
mb1 mb1 ms_ave Ms ms Ms ms1 Ms1"
```

```
# chan_freq_map describes the channel name to frequency
# band mapping in the amplitude table for the FREQUENCY
# DEPENDENT AMPLITUDE BLOCK
chan_freq_map="rms05-1 0.5 1,\
rms05-2 0.5 2,\
rms07-12 0.7 1.2,\
rms07-14 0.7 1.4,\
rms1-2 1.0 2.0,\
rms10-12 10 12,\
rms12-14 12 14,\
rms2-4 2 4,\
rms4-6 4 6,\
rms6-8 6 8,\
rms8-10 8 10"
```

DATABASE

This section describes database elements required for operation of this software component, including accounts, tables, and initialization of the **lastid** table.

Accounts

The only database account required for Retrieve Subsystem operation is the IDCX account as it typically contains all the tables that the Retrieve Subsystem accesses.

Tables

The database account for the Retrieve Subsystem must contain the tables in the Retrieve Subsystem functional group (see [IDC5.1.3Rev0.1]). Table 4 lists Retrieve Subsystem tables and their usage. Figure 6 shows the entity relationships between the major tables of the Retrieve Subsystem (see [IDC5.1.1Rev2]).

TABLE 4: TABLES USED BY THE RETRIEVE SUBSYSTEM

Table	Action	Usage of Attributes
request	reads	all attributes to format request messages to be sent to IMS stations all attributes to compare active requests to data received from IMS stations
	writes	*state* to update the state of requests all attributes to define sample station-channel-time intervals of data to be retrieved from auxiliary stations (polling)
sitepoll	reads	all attributes to obtain information on stations and channels for which sample requests are created
msgdest	writes	all attributes to store information on the destination address, delivery method and time when a request message was sent out
msgdisc	writes	all attributes to create new request messages to be sent by the Message Subsystem to IMS stations
outage	reads	*sta*, *chan*, *time*, *endtime* to obtain information on requests that cannot be satisfied
lastid	reads	*msgid*, *msgdid*, and *reqid* to obtain next available numeric value for these keys
	writes	*msgid*, *msgdid*, and *reqid* to store new last value for these sequential keys
wfdisc	reads	*sta*, *chan*, *time*, *endtime* to compare received data intervals to active waveform requests

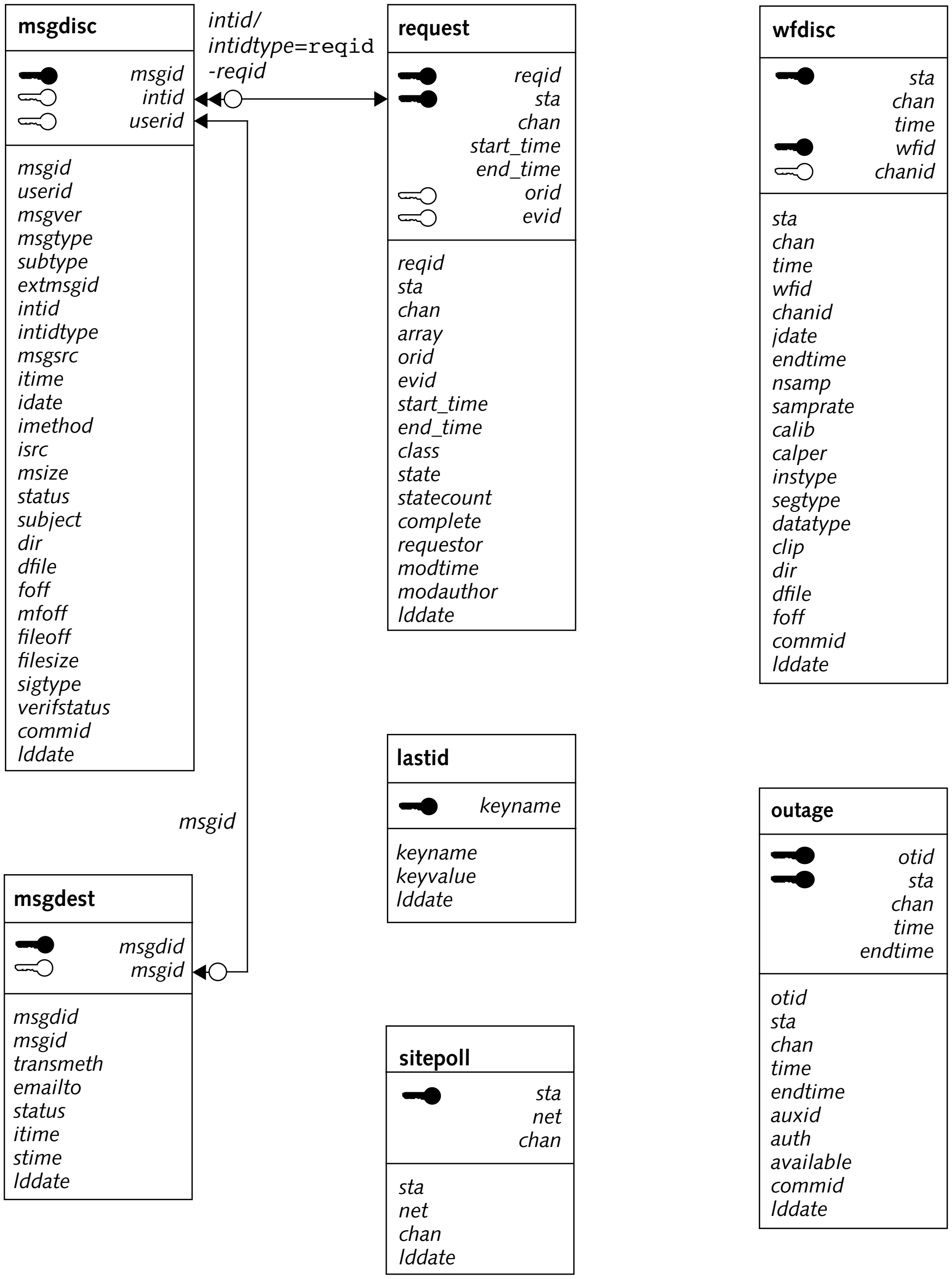


FIGURE 6. RETRIEVE SUBSYSTEM TABLE RELATIONSHIPS

Attributes, variable types and length, and other information needed to create Retrieve Subsystem tables can be found in [IDC5.1.1Rev2] and [IDC5.1.3Rev0.1]. SQL*Plus scripts containing this information are used to create the database tables listed in Table 4. For example, the SQL*Plus script used to create the **request** table follows:

```
create table REQUEST (
        reqid        NUMBER(8)   NOT NULL,
        sta          VARCHAR2(6) NOT NULL,
        chan         VARCHAR2(8) NOT NULL,
        array        VARCHAR2(8),
        orid         NUMBER(8),
        evid         NUMBER(8),
        start_time   FLOAT(53)   NOT NULL,
        end_time     FLOAT(53)   NOT NULL,
        class        VARCHAR2(16),
        state        VARCHAR2(16),
        statecount   NUMBER(8),
        complete     NUMBER(8),
        requestor    VARCHAR2(15),
        modtime      FLOAT(53),
        modauthor    VARCHAR2(15),
        lddate       DATE
) tablespace IDC storage ( initial 80m next 10m
        pctincrease 0 ) pctfree 10;

create unique index REQUESTX on REQUEST(REQID)
   tablespace IDCNDX storage ( initial 10m next 2m
   pctincrease 0 ) pctfree 10 ;

create  index REQENDX on REQUEST(END_TIME)
   tablespace IDCNDX storage ( initial 20m next 5m
   pctincrease 0 ) pctfree 0 ;

create  index REQMODX on REQUEST(MODTIME)
   tablespace IDCNDX storage ( initial 20m next 5m
   pctincrease 0 ) pctfree 0 ;
```

```
grant SELECT on REQUEST to PUBLIC with grant option;
grant DELETE on REQUEST to SEL1 ;
grant INSERT on REQUEST to SEL1 ;
grant UPDATE on REQUEST to SEL1 ;
grant DELETE on REQUEST to SEL2 ;
grant INSERT on REQUEST to SEL2 ;
grant UPDATE on REQUEST to SEL2 ;
grant DELETE on REQUEST to MIGRATE ;
```

Initialization of lastid

The Retrieve Subsystem uses several attributes from the **lastid** table. The **lastid** table serves as a counter for different types of messages by storing a numerical value for the latest message of each type. The attribute *reqid* is read and incremented by data requestors (for example, *WaveExpert*, *polling*, and so on) each time a new **request** table record is created. The attributes *msgid* and *msgdid* are incremented by the Retrieve Subsystem when it writes new message entries to be distributed by the Message Subsystem. These attributes must be present in the **lastid** table and must be initialized to a positive, non-zero value. An example of these attributes in the **lastid** table follows:

```
KEYNAME           KEYVALUE LDDATE
--------------- ---------- -------------------------
msgdid               48021 19-MAR-01
msgid                95742 19-MAR-01
reqid                83535 19-MAR-01
```

TUXEDO FILES

The *dispatch* program is started by *tuxshell*, which is a component of the DACS. Configuration entries in the `ubb_process.tmpl` file and an established Tuxedo queue space are required for proper operation of the Retrieve Subsystem.

Tuxedo is configured for processing all software subsystems during DACS installation. During DACS configuration, the script *crDacsQueues* is used to create queues for all software subsystems under Tuxedo control. If DACS has been properly installed, no Tuxedo file configuration is necessary for Retrieve Subsystem operation.

To ensure that Tuxedo is configured properly for Retrieve Subsystem operation, verify that there are entries for *WaveGet_server*, that the parameter files for *tuxshell* and *WaveGet* are installed, and that the following necessary entries for *tuxshell* are present in the file `ubb_process.tmpl` and the script *crDacsQueues*:

/cmss/config/system_specs/ ubb_process.tmpl

```
tuxshell        SRVGRP=REQ_PRI          SRVID=800

CLOPT="-s dispatch:tuxshell -o /dev/null -e /dev/null --
 par=/cmss/config/app_config/distributed/tuxshell/request/
tuxshell-dispatch.par"

tuxshell        SRVGRP=REQ_BAK          SRVID=10800

CLOPT="-s dispatch:tuxshell -o /dev/null -e /dev/null --
 par=/cmss/config/app_config/distributed/tuxshell/request/
tuxshell-dispatch.par"

TMQFORWARD      SRVGRP=QM_PRI   SRVID=5800

        CLOPT="-- -i 10 -q dispatch     -t 1300"

TMQFORWARD      SRVGRP=QM_BAK   SRVID=15800

        CLOPT="-- -i 10 -q dispatch     -t 1300"
```

/cmss/scripts/bin/crDacsQueues

```
qcreate dispatch         priority,time top,msgid 2 30 80% 0%
"$CMS_SCRIPTS/bin/mailFullQ dispatch"

qcreate failed-dispatch  priority,time top,msgid 2 30 80% 0%
"$CMS_SCRIPTS/bin/mailFullQ failed-dispatch"

qcreate done-dispatch    priority,time top,msgid 2 30 80% 0%
"$CMS_SCRIPTS/bin/mailFullQ done-dispatch"
```

If entries needed to create the *dispatch* queues are not present in the *crDacsQueues* script, they must be added. This should not be necessary as the *crDacsQueues* script is released to the IDC with all necessary entries.

Queues can only be created during configuration of the Tuxedo system. To create a new dispatch queue (or queue for any software) the IDC system must halt processing. The existing queues must be erased and then new queues created with the amended *crDacsQueues* script before restarting IDC processing. For more information on configuring Tuxedo, refer to [IDC6.5.2Rev0.1].

INITIATING OPERATIONS

Before attempting to start the Retrieve Subsystem, check that *WaveAlert* is already running. Use the UNIX `ps -ef | grep` command to look for a *WaveAlert* process.

```
% ps -ef| grep WaveAlert
auto 13471 1 0 Mar 20 ? 20:57 WaveAlert
```

WaveAlert is the only Retrieve Subsystem program that runs continuously and it is usually started with the Message Subsystem via the *keep_msg_alive* script. *WaveAlert* should reside and run with the Message Subsystem programs on its host machine. The following *keep_msg_alive* script is located in `/cmss/scripts/bin`:

```
#!/bin/sh
#
auto=‘imspar AUTO‘
if /usr/bin/ps -fu $auto | /usr/bin/grep MessageReceive | \
   /usr/bin/grep -v grep
then :
else    env ‘cat /cmss/config/system_specs/env/global.shenv‘\
        MessageReceive par=/cmss/config/app_config/messages\
            /MessageReceive/MessageReceive.par &
fi
if /usr/bin/ps -fu $auto | /usr/bin/grep MessageGet | \
   /usr/bin/grep -v grep
then :
else    env ‘cat /cmss/config/system_specs/env/global.shenv‘\
        MessageGet par=/cmss/config/app_config/messages\
        /MessageGet/MessageGet.par &
fi
if /usr/bin/ps -fu $auto | /usr/bin/grep MessageShip | \
   /usr/bin/grep -v grep
then :
else    env ‘cat /cmss/config/system_specs/env/global.shenv‘\
        MessageShip par=/cmss/config/app_config/messages\
        /MessageShip/MessageShip.par &
fi
if /usr/bin/ps -fu $auto | /usr/bin/grep WaveAlert | \
   /usr/bin/grep -v grep
then :
else    env ‘cat /cmss/config/system_specs/env/global.shenv‘\
        WaveAlert par=/cmss/config/app_config/messages\
        /WaveAlert/WaveAlert.par &
fi
```

This script is run daily from *cron* to ensure that required processes are running. This script checks for an existing process before starting a new process with its respective parameter file. Add the line in the example below to the `crontab` file.

CAUTION: The Message Subsystem is typically installed before the Retrieve Subsystem and the `crontab` entry and *keep_msg_alive* script may already be configured for use.

```
% crontab -l
#----------Monitor and restart Message Subsystem
#----------applications that have exited
0,20,40 * * * * ( /usr/bin/env '/usr/bin/cat
/cmss/config/system_specs/env/global.shenv'
/cmss/scripts/bin/keep_msg_alive >/dev/null 2>&1)
```

The `crontab` entry and the *keep_msg_alive* script should ensure that a *WaveAlert* process is always running.

To start *WaveAlert* from a command line, copy the execution command from the *keep_msg_alive* script and paste it onto the command line on the Message Subsystem host. This ensures that proper environments and parameter files are used.

To initiate the operation of the Perl script *polling*, a *cron* entry must be created in the `crontab` file. After this entry is created, *cron* automatically runs *polling* daily. An example of the *polling* `crontab` entry at the PIDC follows:

```
#poll aux stations
29 17 * * * ( env 'cat $(GLOBALDIR)/env/global.shenv'
polling par=$(GLOBALDIR)/process.par )
```

After *WaveAlert* is running and the `crontab` entries for the scripts *keep_msg_alive* and *polling* are created, the Retrieve Subsystem runs automatically after the DACS is running and the *WaveGet* and *tuxshell* servers are started. The remaining programs are run as child processes of the DACS *tuxshell* and terminate themselves after processing.

VALIDATING INSTALLATION

Check the proper installation of the Retrieve Subsystem by verifying that requested auxiliary data are present in the **wfdisc** table. The most efficient way to verify Retrieve Subsystem operation is by using the *RequestFlow* GUI to visually inspect the status of requests in the **request** table.

Use *RequestFlow* to ensure that short, daily requests generated by *polling* are processed. The request intervals generated by *polling* are simple to discern from other auxiliary data requests as they are displayed in a straight line down the *RequestFlow* GUI. This is because *polling* queries each station on the list for an identical time interval.

Use *RequestFlow* to ensure that both standard auxiliary data are processed as well as long-period auxiliary data. Long-period auxiliary stations are displayed above standard auxiliary stations within the *RequestFlow* GUI and are designated with an "`LP`" prefix (for example, `LP BORG`).

Refer to "Monitoring" on page 24 for more information on verifying Retrieve Subsystem operation.

References

The following sources supplement or are referenced in the document:

[Gan79] Gane, C., and Sarson, T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

[IDC3.4.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Formats and Protocols for Messages*, *Revision 2*, SAIC-00/3005, PSR-00/TN2829, 2000.

[IDC3.4.2] Science Applications International Corporation, *Formats and Protocols for Continuous Data*, SAIC-98/3005, 1998.

[IDC3.4.3] Science Applications International Corporation, *Formats and Protocols for Continuous Data CD-1.1*, SAIC-00/3026, 2000.

[IDC5.1.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Database Schema, Revision 2*, SAIC-00/3057, PSR-00/TN2830, 2000.

[IDC5.1.3Rev0.1] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Configuration of PIDC Databases,* SAIC-01/3022, PSR-99/TN1114, 2001.

[IDC6.5.2Rev0.1] Science Applications International Corporation, *Distributed Application Control System (DACS) Software User Manual, Revision 0.1*, SAIC-00/3038, 2000.

[IDC6.5.19] Science Applications International Corporation, *Message Subsystem Software Users Manual*, SAIC-00/3001, 2000.

[IDC7.4.2] Science Applications International Corporation, Pacific-Sierra Research, Inc., *Message Subsystem*, SAIC-98/3003, 1998.

[WGB98a] Working Group B, *Initial Draft of the Operational Manual for the International Data Centre.* Preparatory Commission of the Comprehensive Nuclear-Test-Ban Organization, CTBT/PC/V/WGB/TL/44/Rev.2, 1998.

Glossary

A

analyst

Personnel responsible for reviewing and revising the results of automatic processing.

authentication signature

Series of bytes that are unique to a set of data and that are used to verify the authenticity of the data.

authenticate

Verify the authenticity of a string of bits with an authentication signature.

AutoDRM

Automatic Data Request Manager.

C

CD-ROM

Compact Disk–Read Only Memory.

child process

UNIX process created by the *fork* routine. The child process is a snapshot of the parent at the time it called *fork*.

Computer Software Component

Functionally or logically distinct part of a computer software configuration item; possibly an aggregate of two or more software units.

Computer Software Configuration Item

Aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process.

COTS

Commercial-Off-the-Shelf; terminology that designates products such as hardware or software that can be acquired from existing inventory and used without modification.

CSC

See Computer Software Component.

CSCI

See Computer Software Configuration Item.

D

DACS

Distributed Application Control System. This software supports inter-application message passing and process management.

E

email

Electronic mail.

F

filesystem

Named structure containing files in subdirectories. For example, UNIX can support many filesystems; each has a unique name and can be attached (or mounted) anywhere in the existing file structure.

firewall

Software used to protect a computer or computer network from unauthorized access.

FTP

File Transfer Protocol; protocol for transferring files between computers.

G

GUI

Graphical User Interface.

I

ID

Identification; identifier.

IDC

International Data Centre.

IMS

International Monitoring System.

IPC

Interprocess communication. The messaging system by which applications communicate with each other through *libipc* common library functions. See *tuxshell*.

J

jdate

Modified Julian Date. Concatenation of the year and three-digit Julian day of year. For example, the jdate for 07 March, 2000, is 2000067.

Julian date

Increasing count of the number of days since an arbitrary starting date.

K

key

Data string used by authentication software. Typically keys are defined in pairs, public and private. The private key is used to sign data (produce a validation data value), and the public key is used verify data (determine that a validation data value was produced by the private counterpart of the public key).

L

libgdi

Library containing functions for RDBMS access.

M

MB

Megabyte. 1,024 kilobytes.

message type

Kind of message; possible message types include DATA, REQUEST, and SUBSCRIPTION.

O

ORACLE

Vendor of the database management system used at the PIDC and IDC.

P

par

See parameter.

parameter

User-specified token that controls some aspect of an application (for example, database name, threshold value). Most parameters are specified using [*token* = *value*] strings, for example, `dbname=mydata/base@oracle`.

parameter (par) file

ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files are formatted as a list of `[`*token* `=` *value*`]` strings.

PIDC

Prototype International Data Centre.

pipe

Interprocess communication facility provided by the UNIX operating system. Pipes typically are defined in pairs to support data transmission between two processes where each pipe supports a one-way flow of data.

pipeline

1) Flow of data at the IDC from the receipt of communications to the final automated processed data before analyst review. 2) Sequence of IDC processes controlled by the DACS that either produce a specific product (such as a Standard Event List) or perform a general task (such as station processing).

post-location processing

Software that computes various magnitude estimates and selects data to be retrieved from auxiliary stations.

process

Function or set of functions in an application that perform a task.

product

Bulletins, data, and other information collected, produced, and distributed by the IDC.

program

Organized list of instructions that, when executed, causes the computer to behave in a predetermined manner. A program contains a list of variables and a list of statements that tell the computer what to do with the variables.

Q

query

Request for specific data from a database.

R

radionuclide

Pertaining to the technology for detecting radioactive debris from nuclear reactions.

real time

Actual time during which something takes place.

run

(1) Single, usually continuous, execution of a computer program. (2) To execute a computer program.

S

SAIC

Science Applications International Corporation.

script

Small executable program, written with UNIX and other related commands, that does not need to be compiled.

seismic

Pertaining to elastic waves traveling through the earth.

SEL1

Standard Event List 1; S/H/I bulletin created by total automatic analysis of continuous timeseries data. Typically, the list runs about two hours behind real time.

SEL2

Standard Event List 2; S/H/I bulletin created by totally automatic analysis of both continuous data and segments of data specifically down-loaded from stations of the auxiliary seismic network. Typically, the list runs about six hours behind real time.

server

Software module that accepts requests from clients and other servers and returns replies.

shutdown

Action of terminating a server process as a memory-resident task. Shutting down the whole application is equivalent to terminating all specified server processes (admin servers first, application servers second) in the reverse order that they were booted.

Solaris

Name of the operating system used on Sun Microsystems hardware.

SQL

Structured Query Language; a language for manipulating data in a relational database.

States Parties

Treaty user group who will operate their own or cooperative facilities, which may be National Data Centres.

station

Collection of one or more monitoring instruments. Stations can have either one sensor location (for example, BGCA) or a spatially distributed array of sensors (for example, ASAR).

station processing

Processing based on data from a single station.

subsystem

Secondary or subordinate system within the larger system.

T

tar

Tape archive. UNIX command for storing or retrieving files and directories. Also used to describe the file or tape that contains the archived information.

time series

Time ordered sequence of data samples. Typically a waveform or derived from waveforms, such as a beam.

Tuxedo

Transactions for UNIX Extended for Distributed Operations.

tuxshell

Process in the Distributed Processing CSCI used to execute and manage applications. See IPC.

U

UNIX

Trade name of the operating system used by the Sun workstations.

W

WaveExpert

Application in the Automatic Processing CSCI that determines data intervals to request from auxiliary stations.

Web

World Wide Web; a graphics-intensive environment running on top of the Internet.

wfdisc

Waveform description record or table.

WorkFlow

Software that displays the progress of automated processing systems.

Index